HTML code

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Display Sensor Portal</title>
  {% load static %}
  <link rel="stylesheet" href="{% static 'styles.css' %}">
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/chartjs-adapter-date-fns"></script>
</head>

<body>
  <header>
    <h1>Display Sensor Portal</h1>
    <div class="button-group">
      <button onclick="fetchSensors()" class="button-fetch">Fetch Sensors</button>
      <button onclick="fetchMeasurements()" class="button-fetch">Fetch Measurements</button>
      <button onclick="showChart()" class="button-fetch">Show Chart</button>
    </div>
    <div class="search-group">
      <input type="text" id="query" placeholder="Search...">
      <button onclick="searchSensors()">Search Sensors By Type</button>
      <button onclick="searchMeasurements()">Search Measurements By TimeZone</button>
    </div>
```

```
  </header>
  <main>
    <section id="sensors-section">
      <h2>Sensors</h2>
      <ul id="sensors-list"></ul>
    </section>
    <section id="measurements-section">
      <h2>Measurements</h2>
      <ul id="measurements-list"></ul>
    </section>
    <canvas id="myChart" width="400" height="200" style="display:none;"></canvas>
  </main>
  <script src="{% static 'script.js' %}"></script>
</body>


</html>
```

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Display Sensor Portal</title>
  {% load static %}
  <link rel="stylesheet" href="{% static 'styles.css' %}">
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/chartjs-adapter-date-fns"></script>
</head>
```

head section:

- <meta charset="UTF-8">: **Specifies that the character encoding for the webpage is UTF-8, a widely used character set that can display text in almost any language**.

- <meta name="viewport" content="width=device-width, initial-scale=1.0">: **Ensures that the webpage adapts to different screen widths and ratios on mobile devices, providing a better user experience.**

- <title>Display Sensor Portal</title>: **Sets the title of the webpage, which will be displayed on the browser's tab**.
- {% load static %}: **This is part of Django's template language, used to load static files such as CSS stylesheets or JavaScript files**.

- <link rel="stylesheet" href="{% static 'styles.css' %}">: **Links to an external CSS file, styles.css, which is used to add styles to the webpage. The path is generated using Django's {% static %} tag**.

- **<script src="https://cdn.jsdelivr.net/npm/chart.js"></script> & <script src="https://cdn.jsdelivr.net/npm/chartjs-adapter-date-fns"></script>**: **These lines include external JavaScript libraries. Chart.js is used for rendering charts, and chartjs-adapter-date-fns is used for handling date formats in the charts**.

```
14  <body>
15      <header>
16          <h1>Display Sensor Portal</h1>
17          <div class="button-group">
18              <button onclick="fetchSensors()" class="button-fetch">Fetch Sensors</button>
19              <button onclick="fetchMeasurements()" class="button-fetch">Fetch Measurements</button>
20              <button onclick="showChart()" class="button-fetch">Show Chart</button>
21          </div>
22          <div class="search-group">
23              <input type="text" id="query" placeholder="Search...">
24              <button onclick="searchSensors()">Search Sensors By Type</button>
25              <button onclick="searchMeasurements()">Search Measurements By TimeZone</button>
26          </div>
27      </header>
```

body section:

<header>:

- <h1>: **This is the main title of the webpage, "Display Sensor Portal." It serves as the page's header, indicating to users that this is a portal for displaying sensor information. The <h1> tag defines the size of the title, with options ranging from <h1> to <h6> for different sizes.**

- : **This defines a group of buttons used to perform various actions. The onclick attribute of these buttons specifies the JavaScript function that should be called when the button is clicked**:
    - **The Fetch Sensors button triggers the fetchSensors() function to retrieve sensor data**

- **The Fetch Measurements button triggers the fetchMeasurements() function to retrieve measurement data**
- **The Show Chart button triggers the showChart() function to display a chart**

- : This defines a search area containing an input field and two buttons:
  - **Users can enter their query in the input field.**
  - **The Search Sensors By Type button triggers the searchSensors() function to search for sensors by type.**
  - **The Search Measurements By TimeZone button triggers the searchMeasurements() function to search for measurements based on the time zone.**

```
28      <main>
29          <section id="sensors-section">
30              <h2>Sensors</h2>
31              <ul id="sensors-list"></ul>
32          </section>
33          <section id="measurements-section">
34              <h2>Measurements</h2>
35              <ul id="measurements-list"></ul>
36          </section>
37          <canvas id="myChart" width="400" height="200" style="display:none;"></canvas>
38      </main>
39      <script src="{% static 'script.js' %}"></script>
40  </body>
```

<main>:

The main section of the webpage contains the primary content. It includes two sections that structure the information for sensors and measurements, as well as a canvas element for drawing charts.

- <section id="sensors-section">:
  - <h2>Sensors</h2>: **This is the title for the "Sensors" section, indicating that the following content relates to sensor information**.
  - <ul id="sensors-list">
: This is an unordered list (ul) that will be dynamically populated with sensor information. **Each sensor's details will be displayed as a list item (li) within this list.**
- <section id="measurements-section">:

    - <h2>Measurements</h2>: **This is the title for the "Measurements" section, indicating that the following content relates to measurement data**.
    - <ul id="measurements-list">: **This is another unordered list used to display measurement data. The measurement details will be shown as list items within this list.**
- <canvas id="myChart" width="400" height="200" style="display:none;"></canvas>: **This is a canvas element used for drawing charts. It has an ID of myChart, a width of 400 pixels, and a height of 200 pixels.** By default, this element is hidden (display:none), and **it will become visible when the user clicks the "Show Chart" button, displaying the chart on this canvas**.

<script>:

- **The script tag includes an external JavaScript file, script.js, which contains all the logic related to button interactions, such as fetching sensor data, retrieving measurement data, and drawing the charts**.

In summary, **the body section contains the main visible content of the webpage and defines the primary ways in which users interact with the page. The header section is responsible for navigation and user input, while the main section is responsible for displaying data and charts**

CSS File: **The CSS file controls the styling of your webpage, including fonts, colors, and layout. It helps make the HTML look more visually appealing and adds a sense of design to the page.**

```
/* styles.css */

body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    margin: 0;
    padding: 20px;
}


header {
    text-align: center;
```

```
    padding: 20px;
    background-color: #333;
    color: white;
    margin-bottom: 20px;
}

h1 {
    margin: 0;
    font-size: 2em;
}

.button-group,
.search-group {
    margin: 20px 0;
    text-align: center;
}

button {
    background-color: #007BFF;
    border: none;
    color: white;
    padding: 10px 20px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    margin: 5px 2px;
```

```
    cursor: pointer;
    border-radius: 5px;
    transition: background-color 0.3s ease;
}

.button-fetch {
    background-color: #10aaed;
    border: none;
    color: white;
    padding: 10px 20px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    margin: 5px 2px;
    cursor: pointer;
    border-radius: 5px;
    transition: background-color 0.3s ease;
}

button:hover {
    background-color: #0056b3;
}

input[type="text"] {
    width: 300px;
    padding: 10px;
```

```
    border-radius: 5px;
    border: 1px solid #ccc;
    margin-right: 10px;
    font-size: 16px;
}

main {
    max-width: 800px;
    margin: 0 auto;
    background: white;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

section {
    margin-bottom: 20px;
}

h2 {
    font-size: 1.5em;
    border-bottom: 2px solid #333;
    padding-bottom: 5px;
    margin-bottom: 10px;
}

ul {
```

```
    list-style-type: none;
    padding: 0;
}

li {
    padding: 10px;
    border-bottom: 1px solid #ddd;
}

li:last-child {
    border-bottom: none;
}
```

```css
/* styles.css */
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    margin: 0;
    padding: 20px;
}
```

body { ... }: **This sets the styles for the entire webpage body**.

It specifies the font as Arial, and if Arial is not available, it falls back to sans-serif. The background color is set to light gray (#f4f4f4).

The margin: 0; removes the default margin, allowing the content to be flush with the edges of the page. At the same time, padding: 20px; adds 20 pixels of padding between the content and the page edges, ensuring that the content doesn't touch the edges directly and maintains a comfortable visual spacing

```
header {
    text-align: center;
    padding: 20px;
    background-color: #333;
    color: white;
    margin-bottom: 20px;
}

h1 {
    margin: 0;
    font-size: 2em;
}
```

header { ... }: **Sets the styling for the header section of the page.**

text-align: center; centers the header content (including the title) horizontally. padding: 20px; adds 20 pixels of padding around the header content. background-color: #333; sets the header's background color to dark gray (#333).

color: white; changes the font color of the header content to white. margin-bottom: 20px; adds a 20-pixel margin below the header, creating space between it and the content below.

h1 { ... }: **Sets the styling for the header title**.

margin: 0; removes the margin around the title, making it flush with the rest of the header content. font-size: 2em; sets the font size of the title to 2 times the default font size (usually 32px)

Button and Search Box Styles:

text-align center; centers the content horizontally. The color and background-color properties have been changed, and the margin spacing has been adjusted. For buttons, border is set to none, cursor defines the type of cursor shown, and transition adds a smooth animation effect. The border-radius property gives the buttons slightly rounded corners, so they aren't completely rectangular.

The button:hover effect changes the button's color slightly when hovered over. The input field (input) has also been slightly modified, and the ul element represents an unordered list where each li is a list item.

These styles are primarily used to set the appearance of button groups, search box groups, individual buttons, and text input fields. It standardizes the margins for the button and search box groups and centers the content horizontally. All buttons have a blue

background with white text, rounded corners, and padding. They smoothly transition to a darker blue when hovered over. Certain buttons are set to have a light blue background. The text input field is set to a width of 300 pixels, with padding and a rounded border in light gray, and a margin on the right to keep the layout tidy.

Main Content and Layout Styles:

These styles primarily set the appearance of the main content area and the titles of each section. **The main section defines the main content area with a maximum width of 800 pixels, centered horizontally, with a white background, rounded corners, and a shadow effect, giving it a more dimensional look.** **The section element adds a 20-pixel gap between content blocks, while the h2 title is set to 1.5 times the font size, with a 2-pixel thick dark gray solid line below it to enhance the title's hierarchy.**

```
.button-group,
.search-group {
    margin: 20px 0;
    text-align: center;
}

button {
    background-color: #007BFF;
    border: none;
    color: white;
    padding: 10px 20px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    margin: 5px 2px;
    cursor: pointer;
    border-radius: 5px;
    transition: background-color 0.3s ease;
}

.button-fetch {
    background-color: #10aaed;
    border: none;
    color: white;
    padding: 10px 20px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    margin: 5px 2px;
    cursor: pointer;
    border-radius: 5px;
    transition: background-color 0.3s ease;
}

button:hover {
    background-color: #0056b3;
}

input[type="text"] {
    width: 300px;
    padding: 10px;
    border-radius: 5px;
    border: 1px solid #ccc;
    margin-right: 10px;
    font-size: 16px;
}
```

```
main {
    max-width: 800px;
    margin: 0 auto;
    background: white;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

section {
    margin-bottom: 20px;
}

h2 {
    font-size: 1.5em;
    border-bottom: 2px solid #333;
    padding-bottom: 5px;
    margin-bottom: 10px;
}

ul {
    list-style-type: none;
    padding: 0;
}

li {
    padding: 10px;
    border-bottom: 1px solid #ddd;
}

li:last-child {
    border-bottom: none;
}
```

Javascript:

```
console.log('JavaScript file loaded');


let myChart = null;


async function fetchSensors() {
    try {
        const response = await fetch('/api/Sensor/');
        const sensors = await response.json();
        console.log('Sensors:', sensors);
        const sensorsList = document.getElementById('sensors-list');
        sensorsList.innerHTML = '';
        sensors.forEach(sensor => {
            const listItem = document.createElement('li');
            listItem.textContent = `${sensor.Sensor_ID} - ${sensor.Location}`;
            sensorsList.appendChild(listItem);
        });
    } catch (error) {
        console.error('Error fetching sensors:', error);
    }
}


async function fetchMeasurements() {
    try {
        const response = await fetch('/api/Measurement/');
        const measurements = await response.json();
        console.log('Measurements:', measurements);
```

```
        const measurementsList = document.getElementById('measurements-list');
        measurementsList.innerHTML = '';
        measurements.forEach(measurement => {
            const listItem = document.createElement('li');
            listItem.textContent = `Sensor ID: ${measurement.Sensor_ID} -
${measurement.Temperature}°C - ${measurement.Humidity}%`;
            measurementsList.appendChild(listItem);
        });
    } catch (error) {
        console.error('Error fetching measurements:', error);
    }
}

async function searchSensors() {
    const query = document.getElementById('query').value;
    try {
        const response = await fetch(`/api/Sensor/search/?query=${query}`);
        if (!response.ok) {
            throw new Error('Network response was not ok');
        }
        const sensors = await response.json();
        console.log('Search Sensors:', sensors);

        const sensorsList = document.getElementById('sensors-list');
        sensorsList.innerHTML = '';
        sensors.forEach(sensor => {
            const listItem = document.createElement('li');
```

```
            listItem.textContent = `${sensor.Sensor_ID} - ${sensor.Location}`;
            sensorsList.appendChild(listItem);
        });
    } catch (error) {
        console.error('Error searching sensors:', error);
    }
}

async function searchMeasurements() {
    const query = document.getElementById('query').value;
    try {
        const response = await fetch(`/api/Measurement/search/?query=${query}`);
        if (!response.ok) {
            throw new Error('Network response was not ok');
        }
        const measurements = await response.json();
        console.log('Search Measurements:', measurements);

        // Check if measurements is an array before using forEach
        if (!Array.isArray(measurements)) {
            throw new Error('Expected an array of measurements');
        }

        const measurementsList = document.getElementById('measurements-list');
        measurementsList.innerHTML = '';
        measurements.forEach(measurement => {
            const listItem = document.createElement('li');
```

```
            listItem.textContent = `Sensor ID: ${measurement.Sensor_ID} -
${measurement.Temperature}°C - ${measurement.Humidity}%`;
            measurementsList.appendChild(listItem);
        });
    } catch (error) {
        console.error('Error searching measurements:', error);
    }
}

async function showChart() {
    try {
        const response = await fetch('/api/Measurement/');
        const measurements = await response.json();
        console.log('Measurements for chart:', measurements);
        const timestamps = measurements.map(m => new Date(m.Timestamp));  // 确保将时间戳
转换为日期对象
        const temperatures = measurements.map(m => m.Temperature);
        const humidities = measurements.map(m => m.Humidity);

        const ctx = document.getElementById('myChart').getContext('2d');

        if (myChart) {
            myChart.destroy();
        }

        myChart = new Chart(ctx, {
            type: 'line',
```

```
data: {
    labels: timestamps,
    datasets: [
        {
            label: 'Temperature (°C)',
            data: temperatures,
            borderColor: 'rgba(255, 99, 132, 1)',
            backgroundColor: 'rgba(255, 99, 132, 0.2)',
            fill: false,
        },
        {
            label: 'Humidity (%)',
            data: humidities,
            borderColor: 'rgba(54, 162, 235, 1)',
            backgroundColor: 'rgba(54, 162, 235, 0.2)',
            fill: false,
        }
    ]
},
options: {
    responsive: true,
    scales: {
        x: {
            type: 'time',
            time: {
                unit: 'minute'
            }
```

```
            },
            y: {
                beginAtZero: true
            }
        }
    }
});

    document.getElementById('myChart').style.display = 'block';
} catch (error) {
    console.error('Error fetching measurements for chart:', error);
}
}
```

Initialization and basic settings

```
1   console.log('JavaScript file loaded');
2
3   let myChart = null;
```

- console.log('JavaScript file loaded'); **This statement is used to confirm that the JavaScript file has been successfully loaded. It outputs a message in the browser's console**

- let myChart = null; Declares **a global variable myChart with an initial value of null. This variable will be used to store the chart object, which will be created later**

Function to Fetch Sensor Data

```
async function fetchSensors() {
    try {
        const response = await fetch('/api/Sensor/');
        const sensors = await response.json();
        console.log('Sensors:', sensors);
        const sensorsList = document.getElementById('sensors-list');
        sensorsList.innerHTML = '';
        sensors.forEach(sensor => {
            const listItem = document.createElement('li');
            listItem.textContent = `${sensor.Sensor_ID} - ${sensor.Location}`;
            sensorsList.appendChild(listItem);
        });
    } catch (error) {
        console.error('Error fetching sensors:', error);
    }
}
```

- **The fetchSensors function uses async/await syntax for asynchronous operations to request sensor data from the backend API.**
- fetch('/api/Sensor/'): **Sends a GET request to the server at the /api/Sensor/ path using the fetch function to retrieve sensor data**
- response.json(): **Converts the response data into JSON format**
- sensorsList.innerHTML = ''; **Clears the previous sensor list (to avoid duplicates)**
- sensors.forEach(sensor => { ... }): **Iterates over the retrieved sensor data and adds each sensor's information to the list**

Function to Fetch Measurement Data

```
async function fetchMeasurements() {
    try {
        const response = await fetch('/api/Measurement/');
        const measurements = await response.json();
        console.log('Measurements:', measurements);
        const measurementsList = document.getElementById('measurements-list');
        measurementsList.innerHTML = '';
        measurements.forEach(measurement => {
            const listItem = document.createElement('li');
            listItem.textContent = `Sensor ID: ${measurement.Sensor_ID} - ${measurement.Temperature}°C - ${measurement.Humidity}%`;
            measurementsList.appendChild(listItem);
        });
    } catch (error) {
        console.error('Error fetching measurements:', error);
    }
}
```

- Similar to fetchSensors, this function fetches measurement data.
- fetch('/api/Measurement/'): **Sends a GET request to the /api/Measurement/ path**.
- measurements.forEach(measurement => { ... }): **Iterates over the measurement data and displays it on the page**

Function to Search Sensors

```
39  async function searchSensors() {
40      const query = document.getElementById('query').value;
41      try {
42          const response = await fetch(`/api/Sensor/search/?query=${query}`);
43          if (!response.ok) {
44              throw new Error('Network response was not ok');
45          }
46          const sensors = await response.json();
47          console.log('Search Sensors:', sensors);
48
49          const sensorsList = document.getElementById('sensors-list');
50          sensorsList.innerHTML = '';
51          sensors.forEach(sensor => {
52              const listItem = document.createElement('li');
53              listItem.textContent = `${sensor.Sensor_ID} - ${sensor.Location}`;
54              sensorsList.appendChild(listItem);
55          });
56      } catch (error) {
57          console.error('Error searching sensors:', error);
58      }
59  }
```

- The searchSensors function **allows users to search for sensors based on the query string they input**.
- query: **Retrieves the query string input by the user from the search box**.
- fetch(/api/Sensor/search/?query=${query}): **Sends a GET request with the query string appended to the URL to retrieve the search results**.
- if (!response.ok) { ... }: **Checks if the response was successful; if not, it throws an error**.

Function to Search Measurement Data

```
async function searchMeasurements() {
    const query = document.getElementById('query').value;
    try {
        const response = await fetch(`/api/Measurement/search/?query=${query}`);
        if (!response.ok) {
            throw new Error('Network response was not ok');
        }
        const measurements = await response.json();
        console.log('Search Measurements:', measurements);

        // Check if measurements is an array before using forEach
        if (!Array.isArray(measurements)) {
            throw new Error('Expected an array of measurements');
        }

        const measurementsList = document.getElementById('measurements-list');
        measurementsList.innerHTML = '';
        measurements.forEach(measurement => {
            const listItem = document.createElement('li');
            listItem.textContent = `Sensor ID: ${measurement.Sensor_ID} - ${measurement.Temperature}°C - ${measurement.Humidity}%`;
            measurementsList.appendChild(listItem);
        });
    } catch (error) {
        console.error('Error searching measurements:', error);
    }
}
```

- The searchMeasurements function is similar to searchSensors, but it is used to search for measurement data.
- Array.isArray(measurements): **This step checks whether the returned data is an array. If it is not an array, an error is thrown. This ensures that the data format is correct**.

Function to Display the Chart

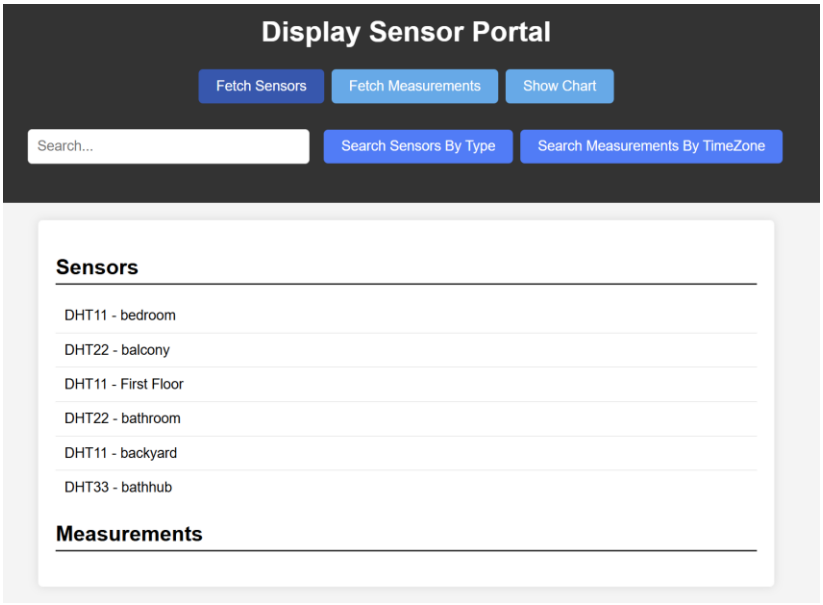
- The showChart function is **used to fetch measurement data and display a chart on the page**.
- timestamps, temperatures, humidities: **Extracts timestamps, temperatures, and humidities from the measurement data and stores them in arrays**

- Chart.js: **Uses the Chart.js library to create a line chart, displaying temperature and humidity data in different colors on the same chart**
- myChart.destroy(); **If a chart has already been drawn, it destroys the previous chart to avoid duplication**

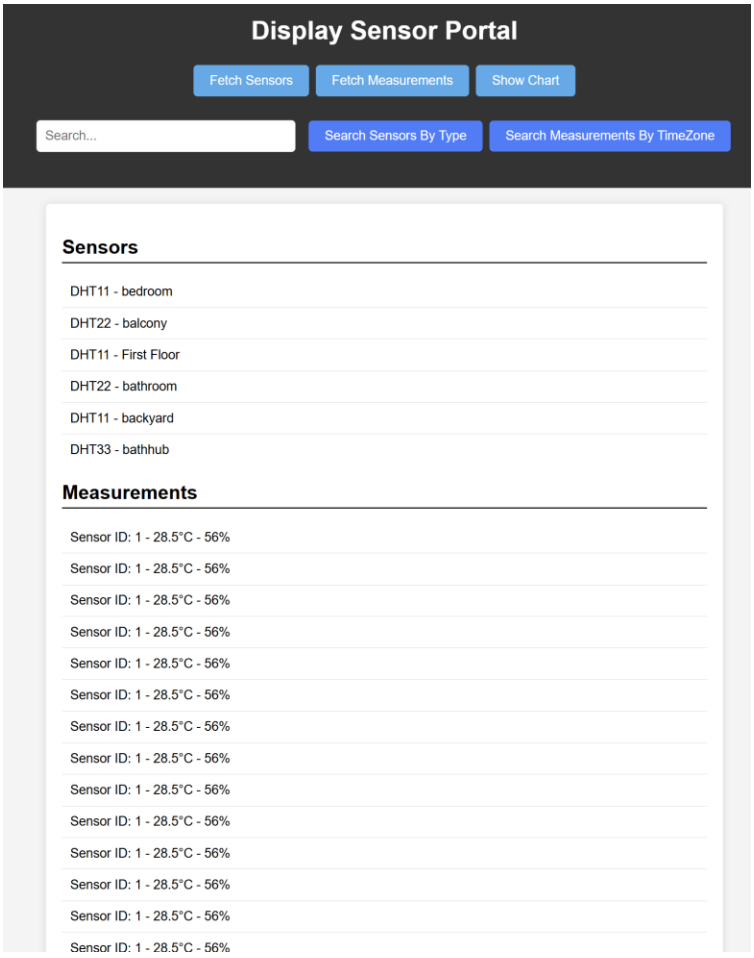
This code primarily interacts with the API to fetch and display sensor and measurement data, utilizing the Chart.js library for data visualization. Each function uses the async/await syntax for asynchronous operations, and the data is dynamically displayed on the webpage through DOM manipulation, enhancing the interactivity of the page.

```
async function showChart() {
    try {
        const response = await fetch('/api/Measurement/');
        const measurements = await response.json();
        console.log('Measurements for chart:', measurements);
        const timestamps = measurements.map(m => new Date(m.Timestamp));
        const temperatures = measurements.map(m => m.Temperature);
        const humidities = measurements.map(m => m.Humidity);

        const ctx = document.getElementById('myChart').getContext('2d');

        if (myChart) {
            myChart.destroy();
        }

        myChart = new Chart(ctx, {
            type: 'line',
            data: {
                labels: timestamps,
                datasets: [
                    {
                        label: 'Temperature (°C)',
                        data: temperatures,
                        borderColor: 'rgba(255, 99, 132, 1)',
                        backgroundColor: 'rgba(255, 99, 132, 0.2)',
                        fill: false,
                    },
                    {
                        label: 'Humidity (%)',
                        data: humidities,
                        borderColor: 'rgba(54, 162, 235, 1)',
                        backgroundColor: 'rgba(54, 162, 235, 0.2)',
                        fill: false,
                    }
                ]
            },
            options: {
                responsive: true,
                scales: {
                    x: {
                        type: 'time',
                        time: {
                            unit: 'minute'
                        }
                    },
                    y: {
                        beginAtZero: true
                    }
                }
            }
        });

        document.getElementById('myChart').style.display = 'block';
    } catch (error) {
        console.error('Error fetching measurements for chart:', error);
    }
}
```

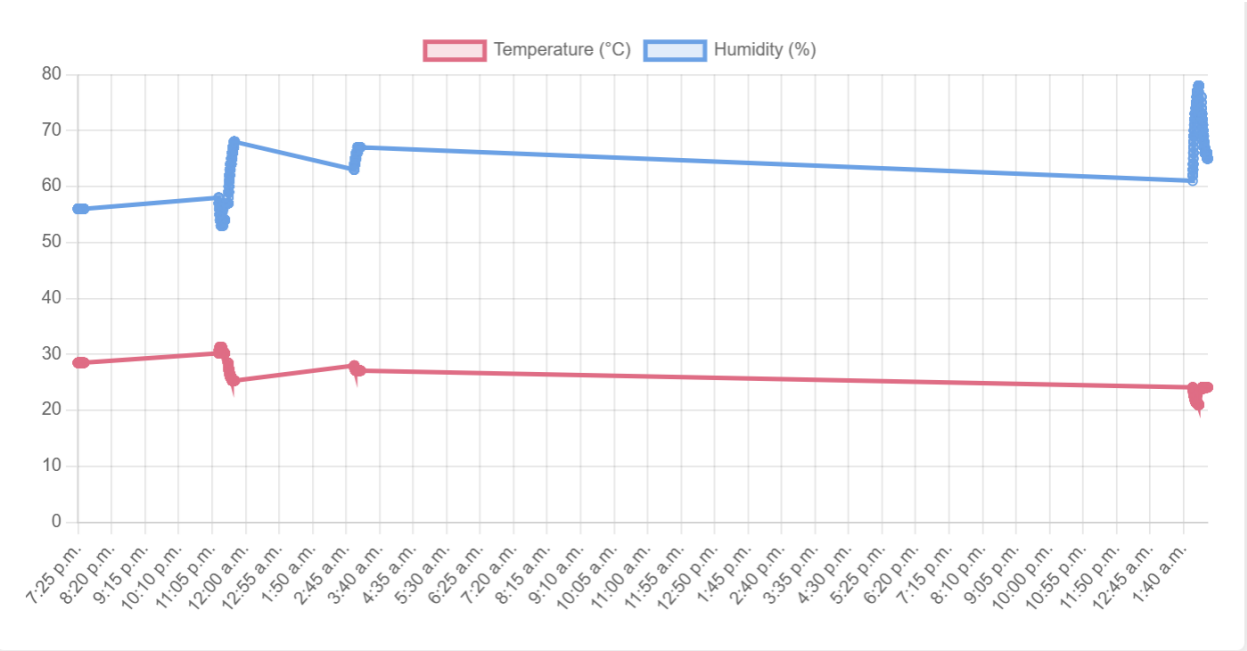
Fetch Sensors



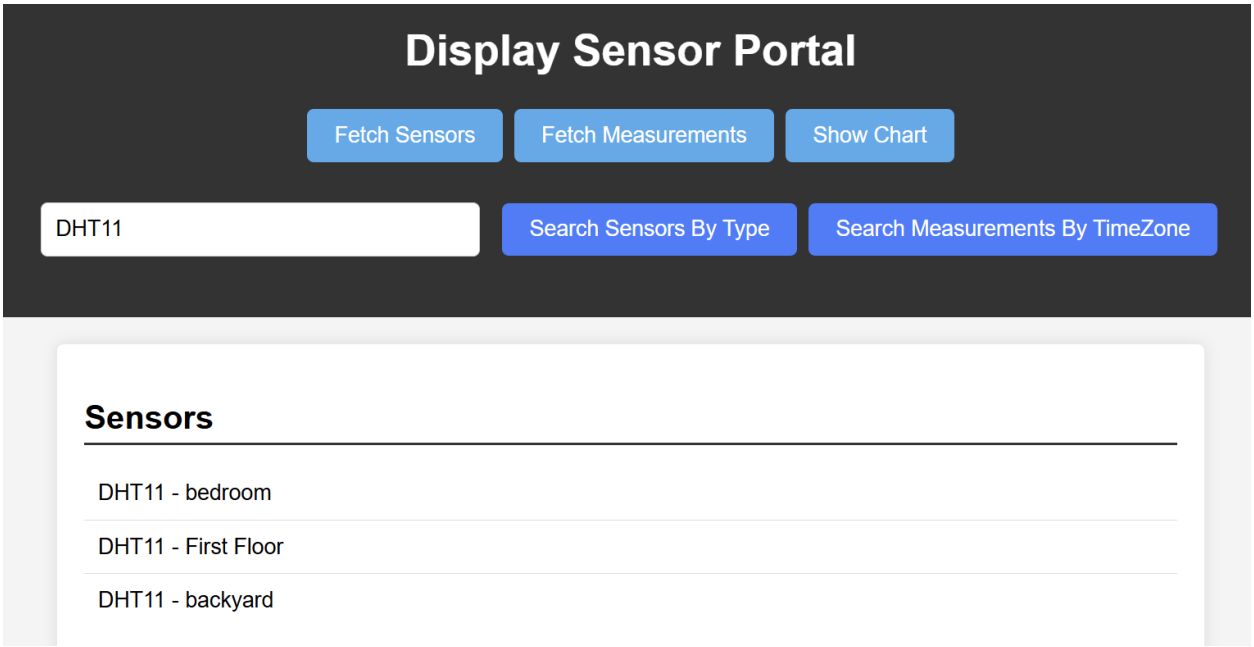
Fetch Measurements



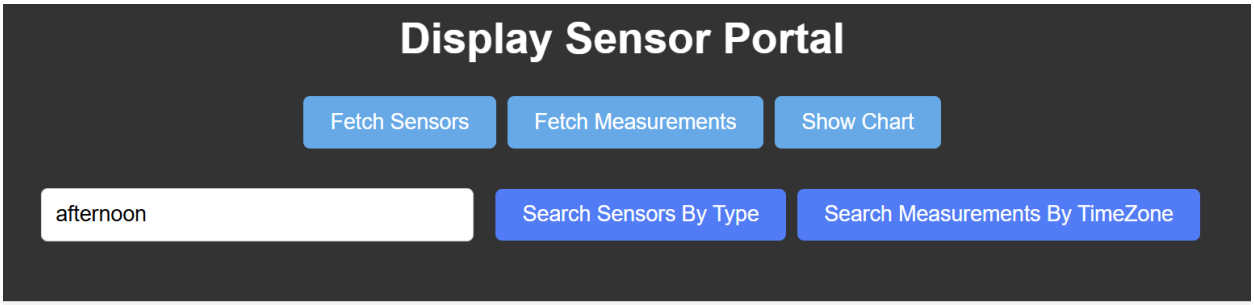
Show Chart



Search Sensors By Type



Search Measurements By TimeZone



Sensors

- DHT11 - bedroom
- DHT11 - First Floor
- DHT11 - backyard

Measurements

- Sensor ID: 2 - 30.2°C - 58%
- Sensor ID: 2 - 30.2°C - 58%
- Sensor ID: 2 - 30.2°C - 58%
- Sensor ID: 2 - 30.2°C - 58%
- Sensor ID: 2 - 30.2°C - 58%
- Sensor ID: 2 - 30.2°C - 58%
- Sensor ID: 2 - 30.2°C - 57%
- Sensor ID: 2 - 30.2°C - 57%
- Sensor ID: 2 - 30.2°C - 57%
- Sensor ID: 2 - 30.2°C - 57%
- Sensor ID: 2 - 30.2°C - 57%
- Sensor ID: 2 - 30.6°C - 57%
- Sensor ID: 2 - 30.8°C - 57%
- Sensor ID: 2 - 30.8°C - 57%
- Sensor ID: 2 - 30.8°C - 57%
- Sensor ID: 2 - 30.8°C - 57%
- Sensor ID: 2 - 30.8°C - 57%

Display Sensor Portal

Fetch Sensors | Fetch Measurements | Show Chart

evening | Search Sensors By Type | Search Measurements By TimeZone

Sensors

- DHT11 - bedroom
- DHT11 - First Floor
- DHT11 - backyard

Measurements

- Sensor ID: 2 - 28°C - 63%
- Sensor ID: 2 - 28°C - 63%
- Sensor ID: 2 - 28°C - 63%
- Sensor ID: 2 - 28°C - 63%
- Sensor ID: 2 - 28°C - 63%
- Sensor ID: 2 - 28°C - 63%
- Sensor ID: 2 - 28°C - 63%
- Sensor ID: 2 - 28°C - 63%
- Sensor ID: 2 - 28°C - 63%
- Sensor ID: 2 - 27.8°C - 64%
- Sensor ID: 2 - 27.6°C - 64%
- Sensor ID: 2 - 27.6°C - 64%
- Sensor ID: 2 - 27.6°C - 64%
- Sensor ID: 2 - 27.6°C - 64%
- Sensor ID: 2 - 27.6°C - 64%
- Sensor ID: 2 - 27.6°C - 64%
- Sensor ID: 2 - 27.6°C - 64%